# A Structured Analysis of SQL Injection Runtime Mitigation Techniques

Stu Steiner
Computer Science Department
University of Idaho
Moscow, ID USA
stusteiner@vandals.uidaho.edu

Daniel Conte de Leon
Computer Science Department
University of Idaho
Moscow, ID USA
dcontedeleon@uidaho.edu

Jim Alves-Foss
Computer Science Department
University of Idaho
Moscow, ID USA
jimaf@uidaho.edu

***Abstract*—SQL injection attacks (SQLIA) still remain one of the most commonly occurring and exploited vulnerabilities. A considerable amount of research concerning SQLIA mitigation techniques has been conducted with the primary resulting solution requiring developers to code defensively. Although, defensive coding is a valid solution, the current market demand for websites is being filled by inexperienced developers with little knowledge of secure development practices. Unlike the successful case of ASLR, no SQLIA runtime mitigation technique has moved from research to enterprise use. This paper presents an in-depth analysis and classification, based on Formal Concept Analysis, of the 10 major SQLIA runtime mitigation techniques. Based on this analysis, one technique was identified that shows the greatest potential for transition to enterprise use. This analysis also serves as an enhanced SQLIA mitigation classification system. Future work includes plans to move the selected SQLIA runtime mitigation technique closer to enterprise use.**

***Index Terms*—SQL; injection, web;**

## I. Introduction

Over the past 15 years, the number of websites has grown from 29 million in 2001 to more than one billion in 2016 [1]. At the beginning of the 21st century the web, known as Web 1.0, consisted of websites that were static pages. Around 2002, Web 2.0 was created and with it came new ideas for exchanging dynamic information. Web 2.0 allows developers to quickly and easily create new dynamic web applications that utilize services and data, stored in back-end databases.

Websites with back-end databases are often susceptible to web vulnerabilities, in particular Structured Query Language (SQL) Injection Attacks (SQLIAs). Over the past 15 years SQLIAs have been actively studied, with various solutions having been proposed; however, no one solution has solved the problem of combating SQLIAs.

Every three years, starting in 2004, the Open Web Application Security Project (OWASP) [2], [3] has published the Top 10 list of web vulnerabilities. In 2004, SQL injection was number six, in 2007, it was number two, and for 2010 and 2013, it remains the number one security risk facing organizations.

Fifteen years ago, Web 1.0 statically linked web pages were typically created by experienced web developers [4]. As Internet usage grew the demand for more dynamic content also grew. With the explosive growth the demand for websites caused a transition from experienced developers to an expanded base of developers with limited knowledge of programming or secure development techniques [5].

Web vulnerability through SQL injection remains a problem for three equally important reasons. First the web is a complicated intertwined set of web pages with millions of lines of unvalidated and unsanitized code. It is simply not cost effective to rewrite millions of lines of code to properly secure it. Second, if an inexperienced developer does not understand SQLIAs, then preventative techniques will not be properly coded into their applications. Third, simple open-source and free tools that can be applied to the millions of lines of insecure code and be used without developer interaction has only been developed in research and has not transitioned to enterprise use. What is urgently needed is a runtime environment that requires little to no programmer involvement, yet still detects a majority of SQLIAs. This type of approach has already been implemented with other security vulnerabilities. A good example is the mitigation of most, but not all, buffer overflow attacks with tools such as StackGuard or Address Space Layout Randomization (ASLR), where both techniques helped raise the security barrier, without requiring major code modifications.

This paper presents an analysis and classification system for SQL injection runtime mitigation techniques. This analysis is based on Formal Concept Analysis (FCA), and it also serves as a formal classification system. The goal is to identify, from all the researched mitigation techniques, which one mitigation technique is the best candidate for technology transfer into the enterprise. Section 2 explains why SQL injection attacks remain a problem. Section 3 discusses a limited set of previously research SQL injection mitigation techniques. Section 4 describes the current commercial SQL injection mitigation techniques. Section 5 explains the selection of the mitigation techniques. Section 6 describes the Formal Concept Analysis background. Section 7 describes the structured analysis based on the FCA. Section 8 presents related work including the prior survey papers. Section 9 presents the conclusion. Section 10 presents future work.

## II. Why SQL Injection Attacks Remain a Problem

Since SQL injection first appeared on the Top 10 list there have been more than 120 papers written about differ-

URI: http://hdl.handle.net/10125/41505
ISBN: 978-0-9981331-0-2

ent classification systems and mitigation techniques. SQLIAs occur in various ways; however, SQLIAs most commonly occur when malicious user-provided data is passed through the web application as SQL commands, and is executed as SQL code by the backend database. In 2006 Halfond et al. [6] characterized seven types of SQLIAs, based on the goal and the intent of the attacker. Those seven types are tautologies, incorrect queries, union query, piggy-backed queries, stored procedures, inference, and alternate encoding.

Others have extended Halfond's et al. work. Since SQLIAs are initiated through a web page, in 2009 Seixas et al. [7] identified and classified the most common security vulnerabilities in web programming languages. In 2012 Ray and Ligatti [8] formally defined web applications and code-injection attacks. In 2013 Shar and Tan [9] broadly classified SQL injection defenses into three categories, defensive coding, SQL injection vulnerabilities detection, and SQLIA runtime prevention. Shar and Tan [9] state, "The best strategy for combating SQL injection . . . calls for integrating defensive coding practices with both vulnerability detection and runtime attack prevention methods." Defensive coding practices are important; however, most developers are typically novices, with less than five years experience (51%), who are self-taught (69%) [5] and most likely have little to no experience with secure development techniques. Furthermore, of the approximate 120 different SQL injection research mitigation techniques, no technique has technologically transferred to enterprise use.

Shar and Tan's concept of ". . . combating SQL injection, . . . with both vulnerability detection and runtime attack prevention methods [9]" is currently improbable. Secure development teaching is very important; however, implementing secure development practices will potentially take years. An immediate solution is needed, specifically a runtime environment, developed with best solution secure coding techniques, that requires little to no programmer involvement.

## III. SQL Injection Mitigation Techniques

One of the goals of this research is to present a FCA-based analysis and classification system for SQL injection runtime mitigation techniques. To achieve this goal, the following objectives were identified.

1) Identify the current research on SQLIA from the ACM digital library and the IEEE Explore digital library.
2) Classify the current research as SQLIA Detection or SQLIA Prevention as explained in subsection III-A.
3) Identify and select the appropriate mitigation techniques for FCA-based analysis as explained in subsection III-B.

Based on the identified goal and objectives 120 SQLIA mitigation technique research papers, spanning the years from 2004 to 2016, were identified.

### A. Applied Classification Method

Previous research of SQLIA mitigation techniques, broadly classified those techniques into two categories; SQLIA Detection (SQLIAD) and SQLIA Prevention (SQLIAP). Halfond et al. [6] define prevention techniques as ". . . techniques that statically identify vulnerabilities in the code, propose a different development paradigm for applications that generate SQL queries, or add checks to the application to enforce defensive coding best practices." Halfond et al. also define detection techniques as ". . . techniques that detect attacks mostly at runtime." To further clarify, static analysis identifies vulnerabilities in the code during the development by checking the source code for vulnerabilities.

Prevention techniques can be further classified into two categories: static (compile-time) analysis or both static and runtime analysis. For static analysis the mitigation technique involves tools to examine the code and identify the potential SQL injection vulnerabilities. The developer needs to manu-

TABLE I
SUMMARY OF SQL INJECTION RUNTIME MITIGATION TECHNIQUES

| Technique | Brief Technique Summary |
|---|---|
| CANDID [10] | CANDID combats SQL injection via obfuscation and de-obfuscation of SQL commands. SQL injection attacks can be detected based on dynamic verification performed on the obfuscated queries. |
| CSSE [11] | Context-Sensitive String Evaluation uses a modified PHP interpreter to track precise per-character taint information through the system. A context sensitive analysis is used to detect and reject queries. |
| SQLCheck [12] | SQLCheck check queries at runtime for conformity to a model of expected queries. The model is expressed as a grammar that will only accept legal queries. In order to check queries at runtime a key is used to delimit user input. |
| SQLGuard [13] | SQLGuard compares the parse tree of the SQL statement before inclusion of user input with the SQL statement after inclusion of user input. The developer must use a special library. |
| SQLProb [14] | SQLProb extracts user query inputs with a pairwise alignment algorithm to compare user queries against legitimate queries. It then uses a SQL parser to check each extracted input. The query is only sent to the database if the user input is syntactically confined. |
| SQLrand [15] | SQLrand provides a proxy server between the web server and database server is used to deciphering the received queries. The proxy server un-randomizes the SQL queries and then forwards the converted SQL query to the database server for execution. It also hides any database server error messages. |
| WASP [16] | WASP uses positive tainting, and tracking techniques for syntax-aware evaluation of queries string. |
| Header Sanitization [17] | Header Sanitization sanitizes received variables inside HTTP header request methods. The sanitized content is replaced back into the original header field. |
| Network Analyzer [18] | Network Analyzer builds a detection system between the attacker and the web server. This system analyzes headers and payload via "Deep Packet Inspection" of the packet. |
| Web Application Firewall [19] | Web Application analyzes the received variables inside HTTP header request methods. The sanitized content is either rejected or passed to the SQL engine if no SQL injection is detected. |

ally modify the vulnerable code to fix the vulnerability. For example, SQLUnitGen by Shin et al. [20] locates vulnerabilities though automated testing, which generates test reports that require the developer to manually correct the vulnerability. Since the majority of developers are inexperienced and self-taught with little knowledge about secure coding techniques, any techniques that include static analysis of code, or static analysis combined with runtime analysis were not considered.

Tools that are both static analysis and runtime prevention require an input generator that creates automated test cases. Those tests are executed against the existing code and are monitored during execution. The results of the execution is used to identify previously undiscovered vulnerabilities and verify the code, and once the vulnerability is detected the correction requires developer involvement. For example, Kieyzun et al. created Ardilla [21] primarily for testing PHP applications before deployment. Ardilla incorporated symbolic logic execution into randomized test inputs to identify previously undetected SQL injection vulnerabilities. Ardilla was also deployed against existing websites where it was able to identify twenty-three previously unknown SQL injection vulnerabilities.

### B. Mitigation Techniques Selection

If a research technique is to transition to enterprise use, that technique needs to completely be a runtime technique. Of the approximate 120 proposed techniques, ten were selected. Table I displays a brief summary of the 10 techniques, with these techniques chosen based on the following criteria:

- Possibility to transition from research to enterprise use
- Number of citations
- Limited programmer involvement
- Limited additional resources
- Considered to be only runtime techniques
- Number of times a technique is mentioned in previous survey papers [6], [9], [22]–[36]

Furthermore, in order to identify which mitigation techniques and characteristics are important for Formal Concept Analysis for SQLIAs, a case study of similar types of security vulnerabilities was conducted. The most closely related vulnerability are buffer overflow attack mitigation techniques. Similar to SQLIAs, buffer overflow attacks can be easily mitigated by the developer through secure development techniques; however, buffer overflow vulnerabilities still exist. Also, SQLIAs mitigation techniques, similar to buffer overflow mitigation techniques have been actively studied for many years; however, unlike buffer overflow attack mitigation techniques, SQLIAs techniques have not successfully transitioned to enterprise use.

## IV. Commercial SQL Mitigation Products

This paper contends that no single SQLIA runtime mitigation technique has transitioned from research to enterprise use. Examining the commercial market for software security yields a relatively new software product category referred to as runtime application self-protection (RASP). RASP appears to be based on research by Halfond et al. [16]; however, this cannot be confirmed since the enterprise version is a proprietary commercial product. Known RASP products are developed and sold by Veracode, Hewlett Packard, and Hdiv Security.

RASP is built into an application's runtime environment, allowing it the capability to control execution. For example, Hewlett Packard created Application Defender. This RASP product monitors and analyzes the API calls to the common core Java libraries [37]. Hdiv Security [38] developed a free, open source community version of RASP targeted to customers who want threat protection with less stringent security requirements. Hdiv Security uses a Java Web Application Security Framework to control the information flow between the server and the client.

RASP seems like a workable solution similar to StackGuard or ASLR; however, it is not the comprehensive product to solve the SQLIA problem. First, RASP relies on the combination of the runtime application and the use of security best practices. Cobb states, "One approach alone will never be sufficient, so don't forget the importance of best practices . . . [37]." Security best practice is very important; however, it is problematic as previously stated based on the inexperience of most developers. Second, RASP like other techniques is still in the infancy of being researched, and is currently mainly available to large companies. It is estimated that less than 1% of the commercial web products use RASP [37]. Furthermore, RASP can be cost prohibitive to smaller companies. Although Hdiv Security does offer a free open source product it is limited in its capabilities; however, if a company requires greater security then there is an associated cost. Finally, RASP is limited in the development languages, such as Java or .NET.

## V. Formal Concept Analysis Background

Formal Concept Analysis (FCA) is ". . . a mathematical formalism which analyses the data in a context and attempts to extract the concepts embodied within that data [39]." FCA is a method for creating a context hierarchy from a collection of objects and their properties. Each concept in the hierarchy represents the set of objects sharing the same values for a certain set of properties or attributes. A hierarchy is a mathematical concept where a set is ordered. For example, the set of integers is one such mathematical hierarchy. For FCA that ordered set is determined by all objects belonging to a concept, and by the collection of all attributes shared by the object [40].

Context [39] is the triplet $(G, M, I)$ where G represents the objects, M represents the attributes, and I represents the relationship of objects to the attributes $I \subseteq (G \times M)$. Content is a pair of sets defined as $(A \subseteq G, B \subseteq M)$, where A is the set of all objects that have all the attributes in B. B represents the set of all attributes that apply to all objects in A.

The context of objects and attributes is constructed as a two-dimensional array of binary values representing the binary relations between the objects and the attributes. Table II is a simple example illustrating the FCA of animals (objects)

TABLE II
SIMPLE FCA EXAMPLE ILLUSTRATING OBJECT/ATTRIBUTES RELATIONSHIP

| Object Name | Land | Water | Trees |
|---|---|---|---|
| Humans | ✓ | | |
| Frogs | ✓ | ✓ | |
| Monkeys | ✓ | | ✓ |
| Giraffes | ✓ | | |
| Fish | | ✓ | |
| Turtles | ✓ | ✓ | |

and the locations where the animals live (attributes). The rows represent the objects (animals) and the columns represent attributes (the locations).

Concepts are understood as ". . . the basic units of thought formed in dynamic processes within social and cultural environments [40]." Concepts and concept hierarchies, are used to create a mathematical model that allows a lexical relationship between objects, attributes, and the relationships of the objects to the attributes. The lexical relationship indicates that an object has an attribute. A lattice is created using the relationship between objects and attributes and the theory of concepts, which is rooted in philosophy and psychology. Although, there are twelve aspects to the theory of concepts, those aspects can be summarized in the following statements.

- The mathematical notion of a formal context converts to the logical meaning of a domain of interest based on object-attribute-relationships.
- The mathematical order-relationship that a formal concept is less than another formal concept is logically understood as a subconcept-superconcept-relationship.
- The mathematical derivation of a set of formal attributes is logically viewed as the identification of all objects having all attributes of a given attribute collection.
- The labeled line diagram of a concept lattice is logically considered as a hierarchical network linking nodes with object names to nodes with attribute names and thereby establishing conceptual meanings.
- Formal object and attribute implications lead to the recognition of conceptual dependencies within the given domain of interest.

The lattice [40] is a visual representation of all objects that share the given attributes, and the relationship of all attributes shared by the given objects. Figure 1 illustrates the lattice generated by the set of concepts illustrated in Table II.

## VI. STRUCTURED ANALYSIS OF RUNTIME MITIGATIONS

Structured analysis is defined as the process of studying a system in order to identify the system's structure, goals and purposes, as well as create systems and procedures that will achieve the structure, goals and purposes in an efficient manner. As previous stated Formal Concept Analysis is ". . . a mathematical formalism which analyses the data in a context and attempts to extract the concepts embodied within that data [39]." FCA is a method for creating a context hierarchy that identifies the system's structure.

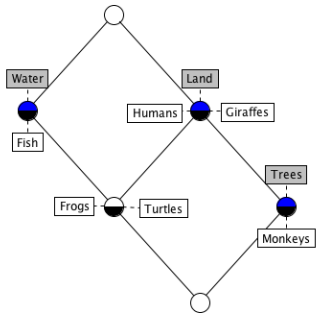


Fig. 1. The lattice for the example FCA. Data represented in Table II

.

### A. Attribute Selection

The attributes were chosen from a combination of attributes in prior survey papers and criteria identified in selecting the 10 mitigation techniques. Not all attributes were chosen from prior survey papers since those attributes did not add value to the FCA. For example a prior survey paper classified the SQLIA mitigation techniques based on the ability to generate test suites for attacks. Since this paper is only considering SQLIA runtime techniques, the ability to generate test suites was not considered when selection attributes. To be considered an attribute from a prior survey paper the attribute had to appear in at least four survey papers. The attributes chosen from the most commonly discussed in the survey papers were:

- classification of the technique as SQL prevention
- classification of the technique as SQL detection
- code base modification
- additional infrastructure requirements
- the types of SQL injection attacks for which the technique would mitigate

Recall the focus of the structured analysis and FCA is to identify which SQL injection runtime mitigation techniques would be the best candidates for technology transfer into the enterprise; therefore, additional attributes needed to be identified. The identified additional attributes that did not appear in any prior survey papers included:

- is the technique language specific
- did the implementation require tracking or tainting
- did the implementation utilize the GET or POST references
- is the technique open source
- is the technique in active development
- what is the level of required developer involvement

In order to be classified as a runtime technique that could transition to enterprise use a majority of all attributes must be met. The attributes that were considered very important were the SQL injection attack types that the technique would mitigate. The reason these SQL injection attack types were classified as very important is because the ultimate goal is to

TABLE III
SQL INJECTION MITIGATION TECHNIQUES FCA

| Mitigation Name | Low Programmer Involvement | High Programmer Involvement | Automatic Code Modification | Manual Code Modification | No Code Modification | Additional Infrastructure | Open Source | Lang. Specific | Active Development | Tracking | GET/POST | Tautology * | Logically Incorrect * | Union Query * | Stored Procedures * | Piggy-Back Query * | Inference * | Alternative Encodings * |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CANDID | ✓ | | ✓ | | | | ✓ | ✓ | | | | ✓ | | | | | | |
| CSSE | ✓ | | | | ✓ | ✓ | ✓ | ✓ | | ✓ | ✓ | ✓ | ✓ | | | | ✓ | |
| SQLCheck | ✓ | | | | ✓ | ✓ | ✓ | | | | | ✓ | ✓ | ✓ | | ✓ | ✓ | ✓ |
| SQLGuard | | ✓ | ✓ | | | | ✓ | ✓ | | | | ✓ | ✓ | ✓ | | ✓ | ✓ | ✓ |
| SQLProb | | ✓ | | | ✓ | | ✓ | | | | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| SQLrand | | ✓ | | ✓ | | ✓ | ✓ | ✓ | | | | ✓ | | ✓ | | ✓ | ✓ | |
| WASP | ✓ | | | | ✓ | | | | ✓ | ✓ | | ✓ | ✓ | ✓ | | ✓ | ✓ | ✓ |
| Header Sanitization | | ✓ | | ✓ | | | ✓ | ✓ | ✓ | | ✓ | ✓ | ✓ | | | | | |
| Network Analyzer | ✓ | | | | ✓ | ✓ | | | ✓ | | ✓ | ✓ | ✓ | ✓ | | | | |
| Web App Firewall | | ✓ | ✓ | | | | | ✓ | ✓ | | ✓ | ✓ | ✓ | ✓ | | | | |

move a product forward raising the security barrier similar to ASLR where SQL injection attacks can be mitigated without developer involvement. Table III illustrates the relationship between the mitigation techniques (objects) and the analysis of each technique (attributes). The attributes includes a mapping of the mitigation techniques to the types of SQL injection and are denoted by the asterisk (*).

### B. Lattice Creation

The free open source software Concept Explorer developed by Sergey Yevtushenko [41] was used to construct a two-dimensional array structure. This two-dimensional array structure is a binary structure represented with ones and zeros, a zero is represented by a blank and a one is represented by a check mark. The objects appear as individual rows and the columns are the individual attributes. The illustration of the objects and attributes is illustrated in Table III.

Using the Concept Explorer software, the two-dimensional array of objects, as mapped to the attributes, was created. The rules are generated by the software which is the set of formal attributes as the identification of all objects having all attributes of a given attribute collection. The rules are listed as antecedents, which precedes the consequence. The rules are based on a percentage of the antecedent supporting the consequence.

### C. Reading the Lattice

A concept lattice is uniquely determined by its formal context, meaning every structural property can be read based on the incidence relation. An incidence relation is defined as the binary relationship between different types of objects, captured by the idea being expressed. For example "a point lies on a line" is an incidence relationship. For FCA lattices the binary relationship is derived from the binary table. Table III clearly illustrates the relationship expressed in the full lattice. Figure 3 illustrates the lattice derived for every structural property based on the incidence relation.

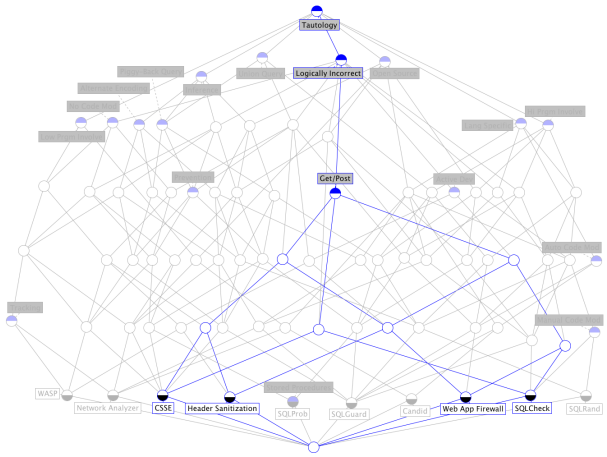


Fig. 2. The Formal Concept Analysis lattice that illustrates the relationship for all objects that contain the attribute GET/POST.

To better understand the lattice and the incidence relations the reader has the option of selecting a single node for any attribute. Selecting such a node will produce a highlighted lattice of the incidence relation of that attribute to the corresponding objects that use that attribute. Figure 2 illustrates the attributes and objects that contain the characteristic GET/POST. The characteristic GET or POST represents the two methods Hypertext Transfer Protocol (HTTP) uses for a request-response between a client and the server. The reader that clicked on the GET/POST node would expect the path to only proceed down towards the bottom of the lattice; however this is often not the case. Recall that every structural property is based on the incidence relation. In Figure 2 the lattice not only proceeds down it also proceeds up. The lattice also proceeds through the attribute logically incorrect to the attribute at the top of the lattice tautology. This relation is based on the objects identified, CSSE, Header Sanitization, Web Application Firewall, and SQL Check; all have the same

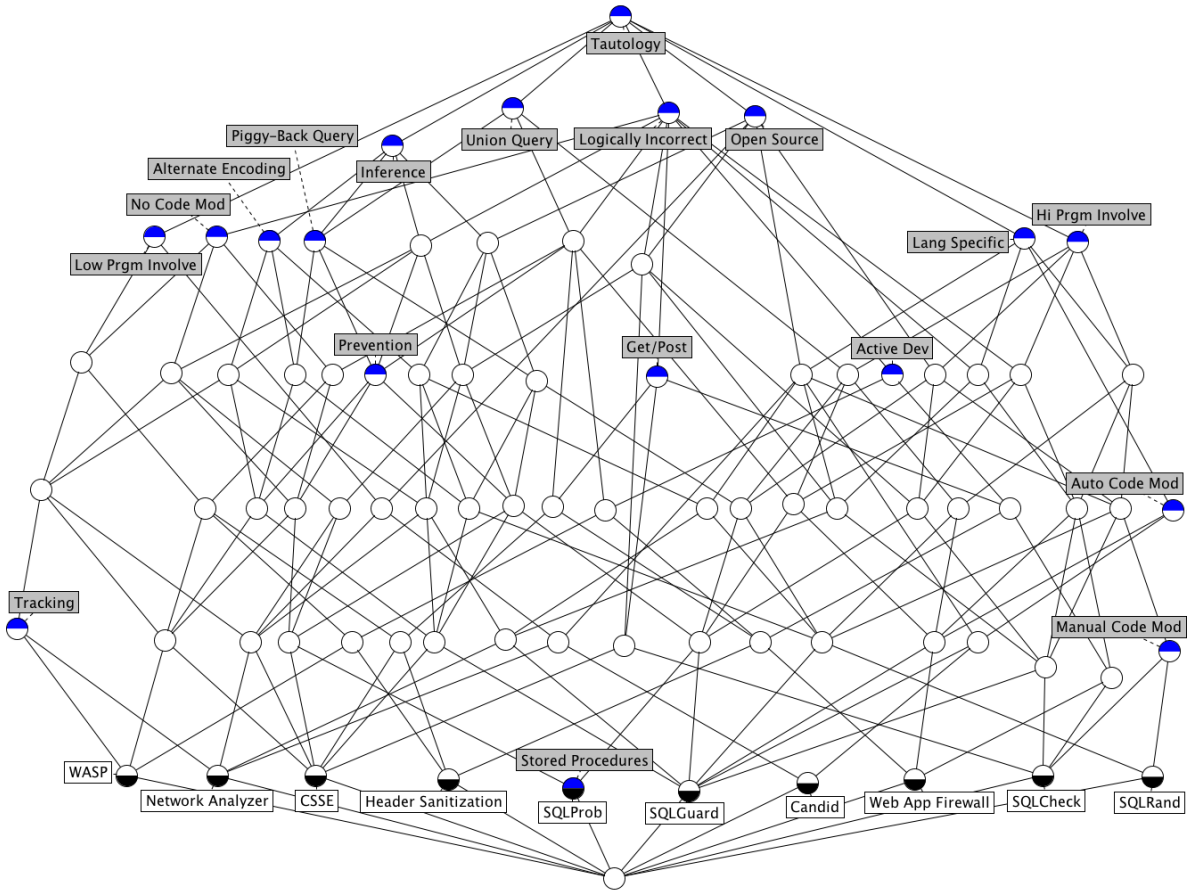


Fig. 3. The lattice for the Formal Concept Analysis

attributes in common including the GET/POST attribute. In order to understand any incidence relations the reader needs to click on any attribute node.

One of the contributions of this paper is identifying the potential runtime mitigation techniques that can be moved to enterprise use. Examining Table III it would appear that SQL Check, SQL Guard, SQL Prob and WASP could all be selected based on the number of SQLIA attack types each one mitigates. SQL Prob combats all the attack types, while SQL Check, SQL Guard and WASP combat all but stored procedures. Since four mitigation techniques cannot all be the best candidates for technology transfer into the enterprise, other incidence relationships must be examined. One of the objectives of this paper is identifying techniques that do not require developer involvement. Selecting high programmer involvement (Hi Prgm Involve) within the lattice would illustrate that SQL Check, SQL Guard and SQL Prob all have high developer involvement for defensively coding against the SQLIAs, while WASP does not share this characteristic. High developer involvement is problematic based on the large number of inexperienced developers. From this comprehensive analysis this paper concludes that WASP is the best candidate tool for evolving into the enterprise. Figure 4 illustrates the incidence relation between the attributes and the technique WASP.

## VII. Related Work

This section describes related work on other classifications of SQLIA mitigation techniques, in addition to the 120 research papers there were 16 papers that were classified as survey papers. To be classified as a survey paper, the contents of the paper had to include the types of SQL injection attacks, the types of vulnerabilities and include an analysis of other mitigation techniques. Since many mitigation technique papers include a summary of the SQL injection types, the summary portion of the paper had to encompass at least one half of the paper.

These 16 papers can be loosely grouped into three categories: papers that classify the mitigation technique to the seven SQLIA types, papers that discuss the strengths and weaknesses of each mitigation technique, including whether the technique is defensive and/or preventive, and papers that discuss the classification of SQLIAs with an analysis of

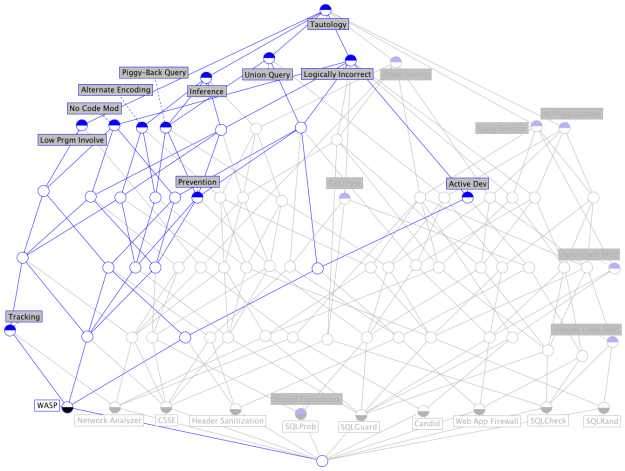


Fig. 4. The Formal Concept Analysis lattice that illustrates WASP [16] is the mitigation technique that should be moved to market.

TABLE IV
SURVEY PAPER CLASSIFICATION WITH PAPER SUMMARY

| Authors | Mitigated SQLIA Types | Strengths & Weaknesses | Risk Analysis | Year |
|---|---|---|---|---|
| Amirtahmasebi et al. [22] | ✓ | | | 2009 |
| Grupta et al. [23] | | ✓ | | 2014 |
| Halfond et al. [6] | ✓ | | | 2006 |
| Johari and Sharma [24] | | ✓ | | 2012 |
| Kaur and Kour [25] | | | ✓ | 2015 |
| Kindy and Pathan [27] | ✓ | | | 2011 |
| Kumar and Pateriya [28] | | ✓ | | 2012 |
| Junjin [29] | | ✓ | | 2009 |
| Mukherjee et al. [30] | | ✓ | | 2015 |
| Sadeghian et al. [31] | | ✓ | | 2013 |
| Sajjadia and Pour [32] | | ✓ | | 2013 |
| Shar and Tan [9] | | ✓ | | 2013 |
| Sharma and Jain [33] | | | ✓ | 2014 |
| Tajpour et al. [34]–[36] | | ✓ | | 2010 |

the risks associated with each attack. Table IV displays the classification of each survey paper. The summary of each paper is as follows:

Amirtahmasebi et al. [22] review the defense mechanisms for six mitigation techniques by discussing very specific details of the defense technique including which SQL injection type the technique protects.

Grupta et al. [23] proposes a classification of the defense techniques of the static analysis based approaches. This survey paper explores eleven techniques from 2005 through 2012.

Halfond et al. [6] classifies the SQL injection attack types. These attack types became the standard attack types papers cite. In this survey paper 17 SQLIA mitigation techniques are compared to the SQL injection attack type, including a classification of the technique as a detection or prevention technique. This paper also includes additional information about modifying the code base and additional infrastructure.

Johari and Sharma [24] present a survey of fourteen prevention techniques that are either SQL injection prevention techniques or cross site scripting prevention techniques. This paper presents a description of each technique. The authors state this ". . . should not excuse developers from applying preventive coding techniques . . ."

Kaur and Kour [25] identify and analyze the various reasons for SQL injection attacks. The paper presents the attack and an example of the attack, but does not present individual mitigation techniques.

Kindy and Pathan's [27] paper provides a detailed review of the various types of SQLIAs including an attempt to classify the individual vulnerabilities into types. These vulnerability types are mapped to the SQLIA types. This paper describes 13 mitigation techniques, including tables mapping each technique to SQL prevention or SQL detection technique, and the SQL injection attack types.

Kumar and Pateriya's [28] paper provides a review of the various types of SQLIAs including an example of each type. The 21 surveyed papers are mapped to the SQLIA types, including mapping the technique to SQL prevention or SQL detection technique, and whether, the technique generates a report.

Junjin [29] proposes an approach for SQL injection vulnerability detection; however, one half of the paper is dedicated to analyzing two other detection techniques. The analysis includes a description of manual approaches and automated approaches for prior SQL injection detection.

Mukherjee et al. [30] provides a review of the SQLIA problem, including the attack type and an example of each attack type. The paper reviews 17 defensive techniques with a classification of each technique as either a prevention or detection technique.

Sadeghian et al. [31] present a review of 15 mitigation techniques. This paper classifies each mitigation as either a best coding practice technique, a detection technique, or a prevention technique.

Sajjadia and Pour [32] provides a taxonomy of prevention and detection techniques. The paper classifies the SQLIA based on the vulnerability type. The paper also addresses prevention techniques as solely static or as hybrid, both static and runtime. This review of eight techniques classifies each technique as prevention or detection, and it includes whether the code base is modified or if there is additional required information for the developer.

Shar and Tan [9] present an analysis of fifteen SQLIA defensive techniques. This paper separates each technique into the categories of defensive coding, detection techniques and runtime techniques, and it reclassifies prevention as runtime techniques and detection as static analysis techniques. This paper states that "Numerous off-the-shelf offerings are useful for quickly detecting the presence of SQLIVs [SQL injection vulnerabilities] in websites." The paper briefly mentions one runtime technique is being commercialized.

Sharma and Jain's [33] paper discusses the classification of SQL injection attacks, including the risk of each attack type. The paper also classifies the vulnerability of each SQLIA, and

discusses the anatomy of orderwise injection types. This paper does not examine any specific defensive technique.

The three papers of Tajpour et al. [34]–[36] present the definition of SQLIAs, and the different attack types, including an example of the attack type. The papers discuss 23 mitigation techniques including mapping the technique to the seven attack types.

This research differs from the other survey papers specifically by the analysis and classification system, based on FCA. None of the 16 survey papers, identified and reported in this section, analyzed almost all reported SQLIA mitigation techniques and their classifications. Also no survey paper discussed which mitigation technique is the best candidate to transition from research to enterprise use. The main objective of this research was to not just repeat prior work. This research provided an analysis and classification of most SQLIA mitigation techniques. This analysis discovered and identified WASP [16] as the most promising technique to technology transfer into the enterprise.

## VIII. Conclusion

SQLIAs are still a problem. This problem has been compounded because most developers are self-taught and inexperienced with little to no knowledge of secure software development techniques. There are many SQLIA mitigation techniques that have been researched; however, no one technique has transitioned to enterprise use. A tool similar to StackGuard or ASLR needs to be implemented in such a manner that it can be utilized without developer involvement. A detailed analysis of all the techniques and the characteristics of the techniques was developed. Based on all the techniques, a classification system was created, and structured Formal Concept Analysis was applied to identify WASP as the best candidate to technology transfer into the enterprise.

## IX. Future Work

Future work includes plans to technologically transfer the one identified SQLIA runtime mitigation techniques closer to enterprise use, by providing an analysis of the current state of the technique.

In order to better understand current secure development practices a survey of the top 10 free web development sites will be conducted. Most free websites allow the user to produce dynamic web pages without knowledge of web programming. Most of these sites contain drag-and-drop tools to aid in the website creation. The survey will include how each free website mitigates SQLIAs, if at all.

Future work includes plans to analyze the possibility of merging WASP with current Open Source Web Application Firewalls (WAF) [42]–[45]. WAF is a technique that is worthy of future research, considering there is little information on how to create a WAF, how to use a WAF, or how to configure a WAF. The combination of WASP and WAF includes the development of a potential market runtime mitigation technique similar to ASLR, as a single "drop in" module to a web application firewall.

## References


[1] NetCraft, "Total number of Websites," 2016.

[2] OWASP, "Category:OWASP Top Ten Project." [Online]. Available: https://www.owasp.org

[3] J. Kravitz, L. Kessem, S. Moore, L. Wiggins, and V. Paliwal, "IBM Security IBM X-Force Threat Intelligence Report 2016," 2016. [Online]. Available: https://developer.ibm.com/identitydev/2016/02/24/ibm-security-ibm-x-force-threat-intelligence-report-2016/

[4] S. Murugesan, "Understanding Web 2.0," *IT Professional*, vol. 9, no. 4, pp. 34–41, jul 2007.

[5] "Stack Overflow Developer Survey Results," 2016. [Online]. Available: http://stackoverflow.com/research/developer-survey-2016

[6] W. G. J. Halfond, J. Viegas, and A. Orso, "A Classification of SQL-Injection Attacks and Countermeasures," in *Proceedings of the IEEE International Symposium on Secure Software Engineering*, Arlington, VA, USA, mar 2006.

[7] N. Seixas, J. Fonseca, M. Vieira, and H. Madeira, "Looking at Web Security Vulnerabilities from the Programming Language Perspective: A Field Study," in *Software Reliability Engineering, 2009. ISSRE '09. 20th International Symposium on*, nov 2009, pp. 129–135.

[8] D. Ray and J. Ligatti, "Defining code-injection attacks," in *Proceedings of the 39th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, ser. POPL '12. New York, NY, USA: ACM, 2012, pp. 179–190.

[9] L. K. Shar and H. B. K. Tan, "Defeating SQL Injection," *Computer*, vol. 46, no. 3, pp. 69–77, 2013.

[10] P. Bisht, P. Madhusudan, and V. N. Venkatakrishnan, "CANDID: Dynamic Candidate Evaluations for Automatic Prevention of SQL Injection Attacks," *ACM Trans. Inf. Syst. Secur.*, vol. 13, no. 2, pp. 14:1–39, mar 2010.

[11] T. Pietraszek and C. V. Berghe, "Defending Against Injection Attacks Through Context-sensitive String Evaluation," in *Proceedings of the 8th International Conference on Recent Advances in Intrusion Detection*, ser. RAID'05. Berlin, Heidelberg: Springer-Verlag, 2006, pp. 124–145.

[12] Z. Su and G. Wassermann, "The Essence of Command Injection Attacks in Web Applications," in *Conference Record of the 33rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, ser. POPL '06. New York, NY, USA: ACM, 2006, pp. 372–382.

[13] G. Buehrer, B. W. Weide, and P. A. G. Sivilotti, "Using Parse Tree Validation to Prevent SQL Injection Attacks," in *Proceedings of the 5th International Workshop on Software Engineering and Middleware*, ser. SEM '05. New York, NY, USA: ACM, 2005, pp. 106–113.

[14] A. Liu, Y. Yuan, D. Wijesekera, and A. Stavrou, "SQLProb: A Proxy-based Architecture Towards Preventing SQL Injection Attacks," in *Proceedings of the 2009 ACM Symposium on Applied Computing*, ser. SAC '09. New York, NY, USA: ACM, 2009, pp. 2054–2061.

[15] S. W. Boyd and A. D. Keromytis, "SQLrand: Preventing SQL Injection Attacks," in *In Proceedings of the 2nd Applied Cryptography and Network Security (ACNS) Conference*, 2004, pp. 292–302.

[16] W. G. J. Halfond, A. Orso, and P. Manolios, "WASP: Protecting Web Applications Using Positive Tainting and Syntax-Aware Evaluation," *Software Engineering, IEEE Transactions on*, vol. 34, no. 1, pp. 65–81, jan 2008.

[17] A. Sadeghian, M. Zamani, and A. A. Manaf, "SQL injection vulnerability general patch using header sanitization," in *Computer, Communications, and Control Technology (I4CT), 2014 International Conference on*, 2014, pp. 239–242.

[18] A. Pramod, A. Ghosh, A. Mohan, M. Shrivastava, and R. Shettar, "SQLI detection system for a safer web application," in *Advance Computing Conference (IACC), 2015 IEEE International*, 2015, pp. 237–240.

[19] A. Makiou, Y. Begriche, and A. Serhrouchni, "Improving Web Application Firewalls to detect advanced SQL injection attacks," in *Information Assurance and Security (IAS), 2014 10th International Conference on*, 2014, pp. 35–40.

[20] Y. Shin, L. Williams, and T. Xie, "SQLUnitGen: Test Case Generation for SQL Injection Detection," Computer Science Dept., North Carolina State University, Tech. Rep., 2006.

[21] A. Kieyzun, P. J. Guo, K. Jayaraman, and M. D. Ernst, "Automatic Creation of SQL Injection and Cross-site Scripting Attacks," in *Proceedings of the 31st International Conference on Software Engineering*, ser. ICSE '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 199–209.

[22] K. Amirtahmasebi, S. R. Jalalinia, and S. Khadem, "A survey of SQL injection defense mechanisms," in *Internet Technology and Secured Transactions, 2009. ICITST 2009. International Conference for*, 2009, pp. 1–8.

[23] M. K. Gupta, M. C. Govil, and G. Singh, "Static analysis approaches to detect SQL injection and cross site scripting vulnerabilities in web applications: A survey," in *Recent Advances and Innovations in Engineering (ICRAIE), 2014*, 2014, pp. 1–5.

[24] R. Johari and P. Sharma, "A Survey on Web Application Vulnerabilities (SQLIA, XSS) Exploitation and Security Engine for SQL Injection," in *Communication Systems and Network Technologies (CSNT), 2012 International Conference on*, 2012, pp. 453–458.

[25] P. Kaur and K. P. Kour, "SQL injection: Study and augmentation," in *Signal Processing, Computing and Control (ISPCC), 2015 International Conference on*, 2015, pp. 102–107.

[26] J. G. Kim, "Injection Attack Detection Using the Removal of SQL Query Attribute Values," in *Information Science and Applications (ICISA), 2011 International Conference on*, 2011, pp. 1–7.

[27] D. A. Kindy and A.-S. Pathan, "A survey on SQL injection: Vulnerabilities, attacks, and prevention techniques," in *Consumer Electronics (ISCE), 2011 IEEE 15th International Symposium on*, jun 2011, pp. 468–471.

[28] P. Kumar and R. K. Pateriya, "A survey on SQL injection attacks, detection and prevention techniques," in *Computing Communication Networking Technologies (ICCCNT), 2012 Third International Conference on*, jul 2012, pp. 1–5.

[29] M. Junjin, "An Approach for SQL Injection Vulnerability Detection," in *Proceedings of the 2009 Sixth International Conference on Information Technology: New Generations*, ser. ITNG '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 1411–1414.

[30] S. Mukherjee, P. Sen, S. Bora, and C. Pradhan, "SQL Injection: A sample review," in *2015 6th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, 2015, pp. 1–7.

[31] A. Sadeghian, M. Zamani, and A. A. Manaf, "A Taxonomy of SQL Injection Detection and Prevention Techniques," in *Informatics and Creative Multimedia (ICICM), 2013 International Conference on*, sep 2013, pp. 53–56.

[32] S. M. S. Sajjadi and B. T. Pour, "Study of SQL Injection Attacks and Countermeasures," *International Journal of Computer and Communication Engineering*, vol. 2, no. 5, 2013.

[33] C. Sharma and S. C. Jain, "Analysis and classification of SQL injection vulnerabilities and attacks on web applications," in *Advances in Engineering and Technology Research (ICAETR), 2014 International Conference on*, 2014, pp. 1–6.

[34] A. Tajpour, M. Massrum, and M. Z. Heydari, "Comparison of SQL injection detection and prevention techniques," in *Education Technology and Computer (ICETC), 2010 2nd International Conference on*, vol. 5, jun 2010, pp. V5———174———V5———179.

[35] A. Tajpour and M. JorJor Zade Shooshtari, "Evaluation of SQL Injection Detection and Prevention Techniques," in *Computational Intelligence, Communication Systems and Networks (CICSyN), 2010 Second International Conference on*, jul 2010, pp. 216–221.

[36] A. Tajpour, M. Z. Heydari, M. Masrom, and S. Ibrahim, "SQL injection detection and prevention tools assessment," in *Computer Science and Information Technology (ICCSIT), 2010 3rd IEEE International Conference on*, vol. 9, 2010, pp. 518–522.

[37] M. Cobb, "Is runtime application self-protection a shortcut to secure software?" Newton MA, pp. 1–12, 2015. [Online]. Available: http://searchsecurity.techtarget.com/opinion/Is-runtime-application-self-protection-a-shortcut-to-secure-software

[38] R. Velasco, G. Vicente, G. Illarramendi, U. Urien, I. Telleria, and A. Berasarte, "Hdiv Community," 2016. [Online]. Available: https://hdivsecurity.com/products.html{\#}community

[39] T. Sutton, "A Complete Idiot's Introduction to Formal Concept Analysis for Dummies to Teach Themselves," GitHub Speaker Deck, Tech. Rep., 2013.

[40] B. Ganter and R. Wille, *Formal Concept Analysis: Mathematical Foundations*, 1st ed. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 1997.

[41] S. A. Yevtushenko, "System of data analysis "Concept Explorer". (In Russian)," in *Proceedings of the 7th National Conference on Artificial Intelligence KII-2000*, 2000, pp. 127–134.

[42] M. Auxilia and D. Tamilselvan, "Anomaly detection using negative security model in web application," in *Computer Information Systems and Industrial Management Applications (CISIM), 2010 International Conference on*, oct 2010, pp. 481–486.

[43] A. Najafi, A. Sepahi, and R. Jalili, "Web driven alert verification," in *Information Security and Cryptology (ISCISC), 2014 11th International ISC Conference on*, sep 2014, pp. 180–185.

[44] A. Tekerek, C. Gemci, and O. F. Bay, "Development of a hybrid web application firewall to prevent web based attacks," in *Application of Information and Communication Technologies (AICT), 2014 IEEE 8th International Conference on*, oct 2014, pp. 1–4.

[45] D. R. Tsai, A. Y. Chang, P. Liu, and H. C. Chen, "Optimum tuning of defense settings for common attacks on the web applications," in *43rd Annual 2009 International Carnahan Conference on Security Technology*, oct 2009, pp. 89–94.